(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 196 22 365 A 1**

(51) Int. Cl.[6]:
G 06 F 5/00
G 06 F 9/44
H 03 M 7/40

(21) Aktenzeichen: 196 22 365.2
(22) Anmeldetag: 4. 6. 96
(43) Offenlegungstag: 11. 12. 97

DE 196 22 365 A 1

(71) Anmelder:
Deutsche Telekom AG, 53113 Bonn, DE

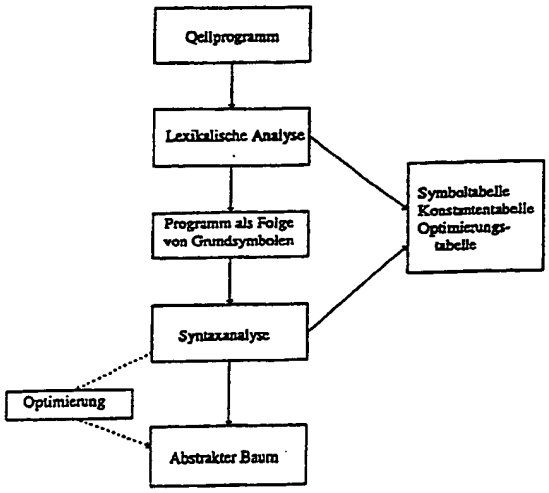
(72) Erfinder:
Knoll, Angelika, Dr., 64293 Darmstadt, DE

(56) Für die Beurteilung der Patentfähigkeit in Betracht zu ziehende Druckschriften:

DE 44 37 790 A1
LANGDON,Glen G.: An Introduction to Arithmetic Coding. In: IBM J. RES. DEVELOP., Vol.28, No.2, March 1984, S.135-149;

(54) Umsetzer zur Übertragung von Quellencode

(57) Es wird ein direkter biteffizienter Umsetzer von Programmcode in einen Bitstrom für die Übertragung angegeben. Die empfängerseitige Rekonstruktion liefert einen syntaktisch und logisch gleichwertigen Programmcode. Dabei wird ein Quellenprogramm, geschrieben in einer Hochsprache wie C/C++, Fortran, Java usw., so in eine Bitfolge umgesetzt, daß zum einen eine möglichst geringe Bitmenge erforderlich ist und zum anderen nach einer Übertragung ein syntaktisch und logisch gleichwertiger Programmcode entsteht. Dazu werden den reservierten Sprachworten, aber auch den Sonderzeichen und Operatoren feste Bitfolgen zugeordnet, die wesentlich weniger Bits benötigen als die direkte Darstellung der reservierten Sprachworte und Operatoren/Sonderzeichen im Rechner. Der Übersetzer kann überall dort angewandt werden, wo Programmcode übertragen werden muß und eine transparente Übertragung nicht erforderlich ist, wie dies zum Beispiel der Fall ist, wenn der übertragene Code im Decoder nur compiliert werden muß und wenn keine weitere Bearbeitung des Codes auf Empfängerseite stattfindet.



DE 196 22 365 A 1

Beschreibung

Die Erfindung betrifft einen Umsetzer zur Übertragung von Quellencode nach dem Oberbegriff des Patentanspruchs 1.

Umsetzer von Programmcode in einen Bitstrom für die Übertragung sind grundsätzlich bekannt. Zum Beispiel muß bei dem in der ISO/IEC Expertengruppe MPEG in der Entwicklung befindlichen Standard MPEG-4 Quellencode bzw. Programmcode mit übertragen werden, um auf Empfängerseite den Decoder zu erzeugen. Auch in anderen Bereichen, in denen Programmcode übertragen werden muß und eine transparente Übertragung nicht erforderlich ist, das heißt der übertragene Code vor der Nutzung im Decoder compiliert werden muß und keine weitere Bearbeitung des Codes auf Empfängerseite stattfindet, kann ein solcher Umsetzer benutzt werden.

Die bekannten Umsetzer von Programmcode in einen Bitstrom für die Übertragung haben jedoch den Nachteil, daß sie nicht biteffizient sind.

Der Erfindung liegt deshalb die Aufgabe zugrunde, einen Umsetzer zur Übertragung von Quellencode dahingehend zu verbessern, daß ein Quellenprogramm, daß in einer Hochsprache geschrieben ist, so in eine Bitfolge umgesetzt wird, daß zum einen eine möglichst geringe Bitmenge erforderlich ist und zum anderen nach einer Übertragung ein syntaktisch und logisch gleichwertiger Programmcode entsteht.

Die erfindungsgemäße Lösung ist im kennzeichnenden Teil des Patentanspruchs 1 charakterisiert.

Weitere Merkmale bzw. Ausgestaltungen ergeben sich aus den an den Patentanspruch 1 anschließenden Patentansprüchen.

Durch diese Maßnahmen wird ein direkter biteffizienter Umsetzer von Programmcode in einen Bitstrom für die Übertragung geschaffen. Die empfängerseitige Rekonstruktion liefert einen syntaktisch und logisch gleichwertigen Programmcode.

Dazu wird der von der Syntaxanalyse erzeugte abstrakte Baum in biteffizienter Weise codiert, wobei den reservierten Sprachworten, aber auch den Sonderzeichen und Operatoren feste Bitfolgen zugeordnet werden, die wesentlich weniger Bits benötigen als die direkte Darstellung der reservierten Sprachworte und Operatoren/ Sonderzeichen im Rechner. Dabei können häufiger vorkommenden Kombinationen eigene Bitfolgen zugeordnet werden.

Besteht ein Programm aus mehreren Modulen, so werden den Funktionsnamen, Klassennamen und Method - namen ebenfalls kurze Bitfolgen zugeordnet; die Länge der Bitfolge hängt von der Anzahl der Namen ab. Sow it diese Originalnamen beim Empfänger bekannt sein müssen, weil zum Beispiel nicht das gesamte Paket übertragen wird, muß diese Tabelle mit übertragen werden. Andernfalls ist dies nicht erforderlich, den entsprechenden Bitfolgen können neue Namen zugeordnet werden. Sind die Originalnamen beim Empfänger bereits bekannt und die entsprechenden Module bereits vorhanden, so ist diese Tabelle ebenfalls erforderlich.

Bei den globalen Variablen, Strukturen und Instanzen werden ähnliche Prinzipien angewandt. Auch hier sind die Originalnamen beim Empfänger meist nicht erforderlich, so daß auf die Übertragung einer Zuordnung von Originalnamen zu Bitfolgen verzichtet werden kann. Ansonsten muß diese Zuordnung in Tabellenform übertragen werden, zumindest für diejenigen Originalnamen, deren Kenntnis im Empfänger erforderlich ist.

Bei lokalen Variablen, Strukturen und Instanzen ist die Übertragung der Zuordnung nicht erforderlich, beim Empfänger können andere Namen gewählt werden.

Die zugewiesenen Bitfolgen sollten so kurz wie möglich gehalten werden. Werden dafür Codes konstant r Länge benutzt, das heißt allen Variablennamen wird die gleiche Länger der Bitfolge zugeordnet, so kann die Länge n der Bitfolge einfach berechnet werden durch Bildung des Logarithmus zur Basis 2 von der entsprech n- den Anzahl und Rundung auf die nächst größere ganze Zahl. Es muß dann entweder die Länge der Bitfolge od r die Anzahl der Variablennamen mit übertragen werden. Bei Verwendung von Codes variabler Länge muß für eine optimale Zuordnung vorher die Häufigkeit des Auftretens bestimmt werden. Der verwendete Code muß dann sender- und empfangsseitig bekannt sein.

Die Darstellung von Konstanten und Zeichenfolgen kann entweder einheitlich über Zeichenketten erfolgen, wobei Zeichenketten in ASCII-Form (mit 7 oder 8 Bit) dargestellt werden. Handelt es sich bei den Konstanten um Zahlen, so wird aus Effizienzgründen in der bevorzugten Realisierung nicht auf die ASCII-Darstellung zurückgegriffen. Integerzahlen werden in dieser Realisierung dabei mit der unten beschriebenen n*(7+1)-Bit Methode codiert, die Zahlen beliebiger Größe verarbeiten kann. Zahlen in Gleitkommadarstellung mit einfach r oder doppelter Genauigkeit werden dabei als 2 Integerzahlen dargestellt, wobei die Integerzahlen entweder als Mantisse und Exponent oder als Vorkomma- und Nachkommawert interpretiert werden.

Diese Bitfolgen werden so gewählt und synataktisch sinnvoll so kombiniert, daß ein eindeutiger und decodierbarer Bitstrom entsteht, der die Rekonstruktion eines syntaktisch und logisch gleichwertigen Programmes erlaubt.

Die Erfindung wird nun anhand eines in den Zeichnungen dargestellten Ausführungsbeispiels näher beschri - ben.

In der Zeichnung ist folgendes dargestellt:

Fig. 1 Phasen bzw. Schritte der Programmübersetzung;

Fig. 2 ein vereinfachtes Beispiel für einen von der Syntaxanalyse erzeugten abstrakten Baum und

Fig. 3 Schritte zur Festlegung des Umsetzers.

Im folgenden wird ein Beispiel für eine Komprimierung gegeben. Hier besteht der komprimierte Bitstrom wesentlich aus Codesymbolen fester Länge. Das Beispiel bezieht sich auf die Programmi rhochsprache C/C++ und beschreibt nicht den vollen Sprachumfang, kann aber einfach auf den vollen Umfang erweitert werden.

Jedes Hochsprachenprogramm besteht aus verschiedenen unterschiedlichen Teilen und Anweisungsarten. Jeder Teil bzw. Anweisungsart beginnt in dieser Realisierung mit einem entsprechenden "Startcode", dessen

Länge von der Anzahl der benötigten Startcodes abhängt. Ein Beispiel der wesentlichsten Anweisungsarten ist in Tabelle 1 gegeben. Für die Beschreibung des vollen Sprachumfanges werden weitere Startcodes benötigt.

Tabelle 1

Startcodes für Programmteile bzw. Anweisungen

#include	00010
#define	00011
Definition globaler Variablen	00100
Definition localer Variablen	00101
Definition einer Methode	00111
Aufruf einer Methode (einer Klasse)	01001
Aufruf einer Methode mit Zuweisung	01010
Instanzierung einer Klasse	01000
Start des "main"-Programmes	01011
Ende des "main"-Programmes oder einer Klasse	01100
Ausführungsanweisung	01101
"while" Anweisung	01110
"end of while"	01111
"for" Anweisung	10000
"end of for"	10001
"if" Anweisung	10010
"else" Anweisung	10011
"end of if"	10100
Definition einer Klasse	10101
Aufruf einer Funktion oder eines Modules	10110

Im folgenden werden Beispiel für einige dieser Programmteile bzw. Anweisungen gegeben.

Darstellung von Integerzahlen

Die folgende Art der Darstellung von Integerzahlen wird in dieser Realisierung verwendet: Das Vorzeichen wird über 1 Bit signalisiert, der Betrag in das duale Zahlensystem gewandelt. Diese Darstellung wird in Stücke der Länge von 7 Bit unterteilt. Die Anzahl der Stücke ergibt die für die Übertragung benötigte Anzahl von Bytes. Das noch freie Bit des Bytes wird mit einer Signalisierung belegt, ob für die Darstellung weitere Bits folgen (n*(7 + 1)-Bit Darstellung).

Der "include" Teil

Werden dem Compiler bekannte Files durch die "include"-Anweisung eingefügt, so kann jedem dieser bekannten Files ein Codesymbol zugeordnet werden. Ein Beispiel hierfür gibt Tabelle 2.

Tabelle 2

Codeworte für bekannte Dateien

<fstream.h>	1000
<string.h>	1001
<iostream.h>	1010
<ctype.h>	1011
<stdio.h>	1100
<stdlib.h>	1101

Der Teil mit "include"-Anweisungen hat in dieser Realisierung folgende Gestalt:

```
#include <fstream.h>
#include <string.h>
#include <iostream.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>,
```

so kann er dann durch die folgende Bitsequenz dargestellt werden, bei der zuerst die binäre Zuordnung für "#include" kommt, dann wird durch ein Bit gekennzeichnet, ob es sich um dem Compiler bekannte Dateien oder um eigene Dateien handelt. Es folgen dann die Anzahl der dem Compiler bekannten einzufügenden Dateien, und dann die Codes für die einzelnen Dateien. Da keine negativen Zahlen auftreten können, entfällt hier di Signalisierung für das Vorzeichen. Somit ergibt sich die folgende Bitfolge:
00010 0 0000110(0) 0110 0001 0010 0011 0100 0101

Definition von globalen/localen Variablen

Die reservierten Sprachworte für elementare Datentypen werden wiederum durch Codes dargestellt. Ein Beispiel hierfür ist in Tabelle 3 gegeben.

Tabelle 3

Reservierte Sprachworte für elementare Datentypen

int	0001		const	1001
char	0010		static	1010
short	0011		signed	1011
long	0100		unsigned	1100
float	0101		*	1101
double	0110		[]	1110
void	0111			

Eine syntaktisch sinnvolle Codierung kann hierbei ähnlich wie im "include"-Teil stattfinden: Zuerst kommt der Startcode, dann die Anzahl der Datentypendefinitionen. In diesem Fall kann der rechte Teil der in Tabelle 3 angegebenen Sprachworte zusätzlich zu den auf der linken Seite der Tabelle angegebenen Sprachworten auftreten. Mit der einfachen Regel, daß zuerst die Codes für die Sprachworte der rechten Seite übertragen werden, kann eine Eindeutigkeit im Bitstrom erreicht werden. Dies bedeutet, daß nach den Codes für "unsigned", "const", "static", "singed", "*" und "[·]" stets noch ein Code für eines der Sprachworte "int", "char", "short", "long", "float", "double", "void" erwartet wird. Bei Feldern wird auch "[·]" auftreten, nach dem Code für die Variablennamen werden noch eine entsprechende Zahl von Integerwerten erwartet. Die Rekonstruktion ist eindeutig.

Die Variablen können einfach durchnumeriert werden, die für die Codierung nötige Bitmenge kann entweder aus der Anzahl der Variablen — sofern diese mit übertragen wird — berechnet werden, oder sie muß mitübertragen werden.

Tabelle 4

Zahl der nötigen Bits zur Darstellung des Variablennamens bzw. des Instanzennamens.

Anzahl der Variablen	Zahl der nötigen Bits zur Darstellung im Bitstrom
2	1
4	2
8	3
16	4
32	5
...	...

Beispiel

```
char*element;
char*buffer[1000];
```

int matrix[10][10];
gibt dann den folgenden Bitstrom:
00100 (Startcode) 0000011(0) (Zahl der Variablen) 1101 ("*") 0010 ("char") 00 (nullte Variable) 1101 ("*") 1110 ("[ ]") 0010 ("char") 01 (erste Variable) 0000111(1)1101000(0) (Feldgröße) 1110 ("[ ]") 1110 ("[ ]") 0001 ("int") 10 (zweite Variable) 000110(0) (Feldgröße) 000110(0) (Feldgröße)
Auf Empfängerseite kann dies dann rekonstruiert werden zu:
char * v0;
char * v1[1000];
int v2[10][10];
mit anderen Namen für die Variablen. Im Decoder müssen diese Namen natürlich in konsistenter Weise verwendet werden.

Beginn des "main"-Programmes

Die Folge: "main() {" kann einfach durch den Startcode wiedergegeben werden. Der entsprechende Code für "end of program" wird dann im Decoder umgesetzt in "};". Bei Übergabe von Argumenten müssen entsprechend Erweiterungen stattfinden.

Instanzenbildung

Die Instanzenbildung von C++ erfolgt analog zur Definition der elementaren Basistypen. Eine eventuell erforderliche Übertragung der Zuordnung von Originalklassennamen zur Darstellung in einer Bitfolge findet vor der Instanzierung statt.

Beispiel

Klassenname *kl_instanz;
ergibt, abgesehen vom Startcode und von der Anzahl der zu bildenden Instanzen,
1101(*) 001 (Bitsequenz für den Klassennamen) 0000 (Instanznr. 0),
wobei angenommen wurde, daß 3 Bit für den Klassennamen und 4 Bit für eine Instanz erforderlich sind.

Definition von Methoden

Wurde zwischen elementaren Datentypen, Strukturen und Klassen unterschieden, das heißt wurden für jeden Typ eigene Bitzuordnungstabellen erstellt, — wie in dieser Realisierung —, so muß nach dem Startcode eine Signalisierung erfolgen, was für ein Datentyp für die Rückgabe vorgesehen ist, andernfalls kann diese Signalisierung entfallen. Der Datentyp für den Rückgabeparameter kann dann entsprechend spezifiziert werden. Werden, wie in dieser Realisierung, die Codeworte für Strukturen und Klassen mit einer "0" begonnen (analog wie in Tabelle 3 die Codeworte für elementare Datentypen), so können auch die zusätzlichen Datentypen der rechten Seite von Tabelle 3 mitverwendet werden, die Eindeutigkeit ist dann garantiert. Es folgt der Code für den Klassennamen, der Code für den Namen der Methode und ein weiteres Bit zur Signalisierung, ob diese Method mit Parametern aufgerufen wird. Wenn ja, kann dann die Anzahl folgen und die Parameter können analog zur Definition der Variablen/Strukturen/Instanzen codiert werden. Hierbei ist natürlich darauf zu achten, daß entweder die benutzte Bitmenge groß genug ist, um klasseninterne Parameter und Argumente damit darstellen zu können, oder es wird eine entsprechende Signalisierung benutzt.

Aufruf einer Methode

Findet nicht gleichzeitig noch eine Zuweisung statt, so wird zuerst der Startcode kommen, dann die Bits für die Klasseninstanz. Der sich anschließende "." oder "→" kann mit einem Signalisierungsbit übertragen werden. Es folgt dann die Bitfolge für die Methode. Im allgemeinen ist die Anzahl der Argumente nicht bekannt, es folgt also eine Signalisierung, ob und wenn ja, wie viele Argumente vorhanden sind. Diese werden dann ebenfalls in Bitfolgen umgesetzt. Haben Variable, Strukturen, Instanzen und Methoden/Funktionen keine einheitliche Bitfolgenzuordnung, so muß, — wenn die Definition der Funktion/Methode nicht bekannt ist und damit der Datentyp dem Decoder nicht bekannt ist —, jeweils durch zum Beispiel 2 Bit signalisiert werden, um welchen Typ es sich bei dem jeweiligen Argument handelt.

Ausführungsanweisungen

Die meisten Ausführungsanweisungen sind recht kurz, und oft ist die Folge von Variablen/Strukturen/Instanzen (v), Sonderzeichen/Operatoren (o) und Konstanten/Strings (c) häufig vorhanden, so daß es sinnvoll ist, für häufig vorkommende derartige Folgen einen eigenen Code zu verwenden. Dabei können auch spezielle Anweisungen wie zum Beispiel "variable++;" oder "variable=0;" einen eigenen Code erhalten. Ein kurzes Beispiel einer derartigen Zuordnung ist in der folgenden Tabelle gegeben. Vor der Codierung der Konstanten/Strings wird eine Signalisierung gesetzt, die angibt, um was es sich im folgenden handelt.

Tabelle 7

Beispiel für die Zuordnung einiger häufiger Folgen von Variablen/Strukturen/Instanzen (v), Sonderzeichen/Operatoren (o) und Konstanten/Strings zu Bitfolgen

vovov	00001	
ov	00010	
vo	00011	
vov	00100	
vovovo	00101	
voc	00110	
vovoc	00111	
vp	01000	var++
vz	01001	var=0
vi	01010	var=integer
leer	00000	
escape	11111	
escapedg	11110	var[...]=...

Der in der folgenden Tabelle angegebene Code für das Sonderzeichen ";" tritt nur in Verbindung mit "escape" auf. Ist die Folge von Variablen/Strukturen/Instanzen (v), Sonderzeichen/Operatoren (o) und Konstanten/Strings (c) bekannt, so ist das Sonderzeichen ";" nicht erforderlich.

Tabelle 8

Beispiel für die Zuordnung der Sonderzeichen zu Bitfolgen

+	00001		++	00101
-	00010		--	00110
*	00011		+=	00111
/	00100		-=	01000
!	01010		&&	01001
<	01111		!=	01011
>	10001		or	01100
[	10010		<<	01101
]	10011		>>	01110

=	10100		<=	10101
&	11001		>=	10110
(	11010		*=	10111
)	11011		&=	11000
;	11111		((	11100
			))	11101
			]=	11110

Im folgenden Beispiel wird der Startcode nicht mit aufgeführt, vier Bits seien für die Darstellung von Variablen nötig. Die Unterscheidung zwischen Variablen, Strukturen, Instanzen und Methoden/Funktionsaufrufen geschieht hier mit einem 2-Bit-Flag.

i++;	01000 (vp) 01 (Variable folgt) 0010 (zweite Variable)
loop=0;	01001 (vz) 0011 (dritte Variable)
buffer[i]='2';	00111 (vovoc) 01(Variable folgt) 0001 (erste Variable) 10010 (Sonderzeichen:[) 01 (Variable folgt) 0010 (zweite Variable) 11110(Sonderzeichen:]=) 00000110 (3 Byte ASCII folgen) '2' (in ASCII übertragen)
buf=buffer[i];	00101 (vovovo) 01 (Variable folgt) 0100 (vierte Variable) 10100 (Sonderzeichen:=) 01 (Variable folgt) 0001 (erste Variable) 10010 (Sonderzeichen:[) 01 (Variable folgt) 0010 (zweite Variable) 10011 (Sonderzeichen:])

Um komplexeren Anweisungen, deren Reihenfolge nicht in der obigen Tabelle aufgenommen ist, Bitfolgen zuzuweisen, kann "escape" mit einer modifizierten Syntaxstruktur benutzt werden, deren Struktur wohl am besten aus dem folgenden Beispiel hervorgeht. Durch entsprechende Signalisierung wird dabei stets angegeben, was als nächstes zu erwarten ist:
Variable/Struktur/Instanz
Sonderzeichen/Operator
Funktion/Methodenaufruf
In Fällen, in denen diese Signalisierung nicht ausreicht, folgt eine zweite Signalisierung.

Beispiel

Der Anweisung

buffer[i]=((i+k)/(buf−1))&&buffer[k];

kann dann, abgesehen wiederum vom Startcode, die folgende Bitfolge zugeordnet werden:
11111 (escape) 01 (Variable/Struktur/Instanz folgt) 01 (Variable folgt) 0001 (erste Variable) 10 (Sonderzeichen/Operator folgt) 10010 (Sonderzeichen: [) 01 (Variable/Struktur/Instanz folgt) 01 (Variable folgt) 0010 (zweite Variable) 10 (Sonderzeichen/Operator folgt) 11110 (Sonderzeichen:]=) 10 (Sonderzeichen/Operator folgt) 11111 (Code für Befehlsende:) usw.

Die Benutzung von "escape" benötigt im allgemeinen eine größere Bitmenge als wenn eine der vordefinierten Folgen von Variablen/Strukturen/Instanzen (v) Sonderzeichen/Operatoren (o) und Konstanten/Strings (c) benutzt wird.

Kontrollanweisungen ("for", "while", "if", "else") "for"

Nach dem Sprachwort "for" werden 3 Ausdrücke, getrennt durch ";" in Klammern erwartet, wobei in diesen Ausdrücken auch Kommas erlaubt sind. Nach dem Startcode für die "for"-Anweisung folgen direkt 3 Ausführungsanweisungen, die natürlich auch leer sein können. Auf Empfängerseite wird dann die entsprechende Syntax mit "for(.. ; .. ; ..) {" wiederhergestellt. Für das Ende der Schleife wird auch in entsprechender "Startcode" übertragen. Dazwischen können beliebige Anweisungen erfolgen.

Im folgenden Beispiel ist "k" die 14. Variable und "element" die 1. Variable. Die Anweisung

for (k = 0; k < element; k + +){

kann dann die folgende Bitfolge geben:
10000 (start code)
erster Ausdruck:
01001 (vz) 01 (Variable folgt) 1110 (14. Variable)
zweiter Ausdruck:
00100 (vov) 01 (Variable folgt) 1110 (14. Variable) 01111 (Sonderzeichen: <) 01 (Variable folgt) 0001 (1. Variable)
dritter Ausdruck:
01000 (vp) 01 (Variable folgt) 1110 (14. Variable)

"while"

Der Unterschied zur "for"-Anweisung besteht ausschließlich in der Anzahl der in Klammern nachfolgenden Ausdrücke, die Syntax kann also analog erfolgen.

"if"

Analog zur "while"-Anweisung erwartet die "if"-Anweisung ebenfalls einem Ausdruck, die Syntax erfolgt analog zu "while". Der Startcode für das Ende der "if"-Schleife wird dann im Decoder umgesetzt in "};", der Cod für die "else"-Anweisung wird im Empfänger wieder umgesetzt in "} else {", so daß vor der "else"-Anweisung kein Code zu erfolgen hat, der das Ende der "if"-Anweisung gibt. Die "elseif"-Anweisung muß ebenfalls "}" mit erzeugen, erwartet ansonsten aber auch einen Ausdruck.

Patentansprüche

1. Umsetzer zur Übertragung von Quellencode, insbesondere einer Hochsprache wie zum Beispiel C/C++, Fortran oder Java in einen Bitstrom für die Übertragung, wobei die empfängerseitige Rekonstruktion einen syntaktisch und logisch gleichwertigen Programmcode liefert, **dadurch gekennzeichnet,**
daß ein mittels Syntaxanalyse erzeugter abstrakter Baum in biteffizienter Weise codiert wird und
daß zur Festlegung des Umsetzers reservierten oder speziellen Sprachworten kurze Bitfolgen zugeordnet werden,
daß mittels lexikalischer Analyse den Symbolen kurze Bitfolgen zugeordnet werden,
daß aufgrund der lexikalischen Analyse auch Konstanten in Bitfolgen umgesetzt werden,
daß Anweisungen wie Deklaration, Funktionsaufruf, arithmetische Anweisung usw., Schlüsselworte als kurze Bitfolgen zugeordnet werden,
daß die Struktur eines jeweiligen Teilbaumes angegeben wird, falls diese Struktur nicht eindeutig mit dem Schlüsselwort festgelegt ist,
daß Schlüsselworte und Bitfolgen in eindeutiger Weise kombiniert werden und
daß für Symbole, die nach außen bekannt sein müssen, die Zuordnung des Symbols zu der intern benutzten Bitfolge mit übertragen wird.
2. Verfahren nach Patentanspruch 1, dadurch gekennzeichnet, daß bei Verwendung von variablen Längencodes für die Symbole, zum Beispiel bei Huffmann oder Entropiecodierung, die Häufigkeit jedes Symb les bestimmt wird und eine entsprechende Zuordnung zu Bitfolgen möglichst geringer Länge erfolgt und/oder für die reservierten Sprachworte (inklusive Sonderzeichen und Operatoren) die Häufigkeit jedes reservierten Sprachwortes bestimmt wird und eine entsprechende Zuordnung zu Bitfolgen möglichst geringer Länge erfolgt.
3. Verfahren nach Patentanspruch 1, dadurch gekennzeichnet, daß bei Verwendung fester Längencodes für die Symbole diesen entsprechende Bitfolgen zugeordnet werden, wobei die Länge der Bitfolge von der Anzahl der Symbole abhängt und die Länge oder die Anzahl mit übertragen wird und/oder für die reservierten Sprachworte (inklusive Sonderzeichen und Operatoren) diesen feste Bitfolgen zugeordnet werden, die wesentlich weniger Bits benötigen als die direkte Darstellung.
4. Verfahren nach Patentanspruch 2 oder 3, dadurch gekennzeichnet, daß für alle Symbole über eine einheitliche Tabelle eine Zuordnung von Bitfolgen erfolgt.
5. Verfahren nach Patentanspruch 2 oder 3, dadurch gekennzeichnet, daß für jede Symbolart wie zum Beispiel Variablenname, Funktionsname, Klassenname, Instanz eine eigene Symboltabelle erstellt wird und damit die Zuordnung von Bitfolgen getrennt erfolgt und daß in nicht eindeutigen Fällen die Symbolart mit übertragen wird.
6. Verfahren nach Patentanspruch 2 oder 3, dadurch gekennzeichnet, daß häufiger vorkommenden Kombinationen eigene Bitfolgen zugeordnet werden.
7. Verfahren nach Patentanspruch 1, dadurch gekennzeichnet, daß die Zuordnung von Originalsymbolnamen zu den Bitfolgen dann zusätzlich übertragen wird, wenn diese beim Empfänger bekannt sein müssen.
8. Verfahren nach Patentanspruch 3, dadurch gekennzeichnet, daß bei Verwendung fester Längencodes die Länge der Bitfolge berechnet wird durch Bildung des Logarithmus zur Basis 2 von der entsprechenden Anzahl und Rundung auf die nächst größere ganze Zahl, wob i dann ntweder die Länge der Bitfolge oder die Anzahl der variablen Namen mit übertragen wird.
9. Verfahren nach einem der Patentansprüche 1 bis 8, dadurch gekennzeichnet,
daß die Darstellung von Konstanten und Zeichenfolgen einheitlich über Zeichenketten erfolgt, wobei die

Zeichenketten in ASCII-Form dargestellt werden und
daß Integerzahlen mit der n*(7 + 1)-Bitmethode codiert werden, wobei Zahlen in Gleitkommadarstellung mit einfacher oder doppelter Genauigkeit als zwei Integerzahlen dargestellt werden und die Integerzahlen entweder als Mantisse und Exponent oder als Vorkomma- und Nachkommawert interpretiert werden.

Hierzu 3 Seite(n) Zeichnungen

Nummer: DE 196 22 365 A1
Int. Cl.[6]: G 06 F 5/00
Offenlegungstag: 11. Dezember 1997

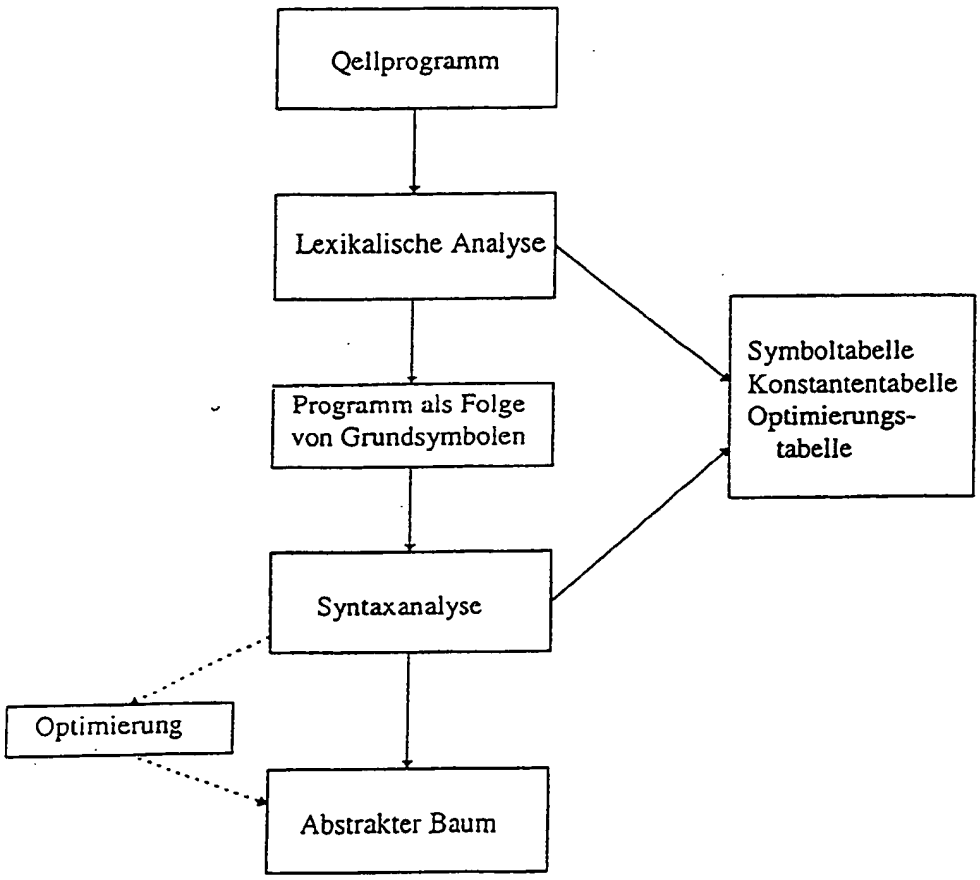


FIG. 1

Nummer: DE 196 22 365 A1
Int. Cl.[6]: G 06 F 5/00
Offenlegungstag: 11. Dezember 1997

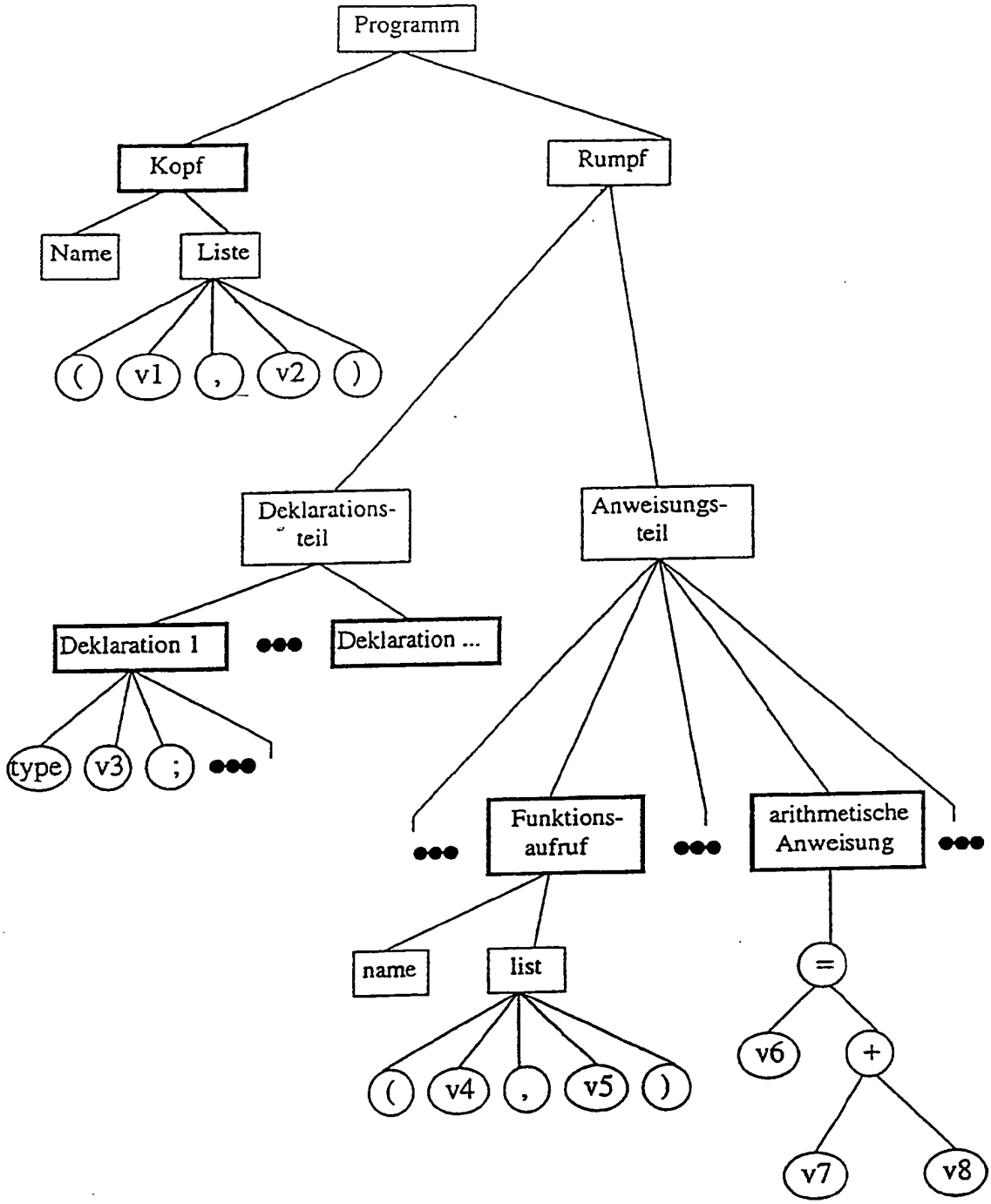


FIG. 2

Nummer: DE 196 22 365 A1
Int. Cl.[6]: G 06 F 5/00
Offenlegungstag: 11. Dezember 1997

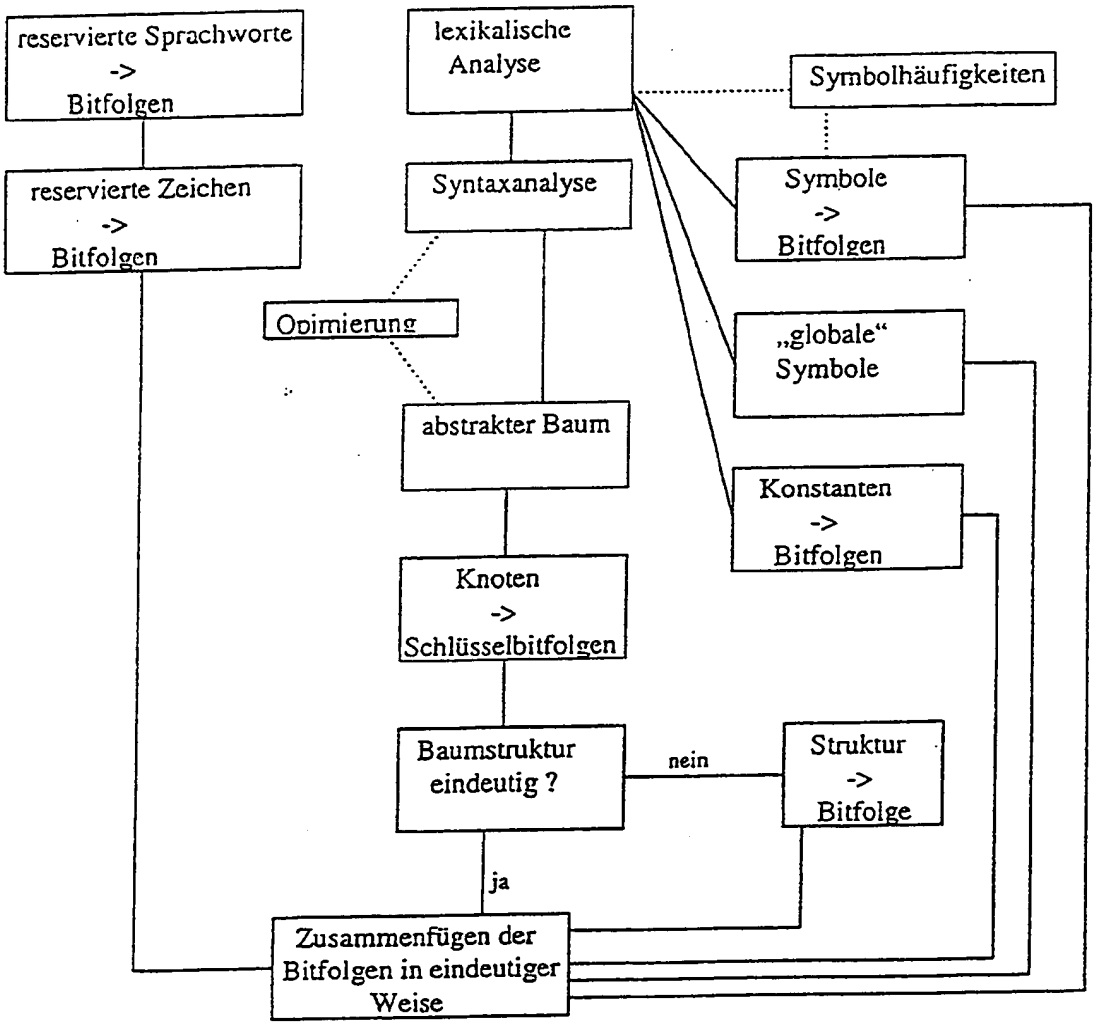


FIG. 3

Nummer: DE 199 22 365 A1
Int. Cl.[5]: G 06 F 8/00
Offenlegungstag: 11. Dezember 1997

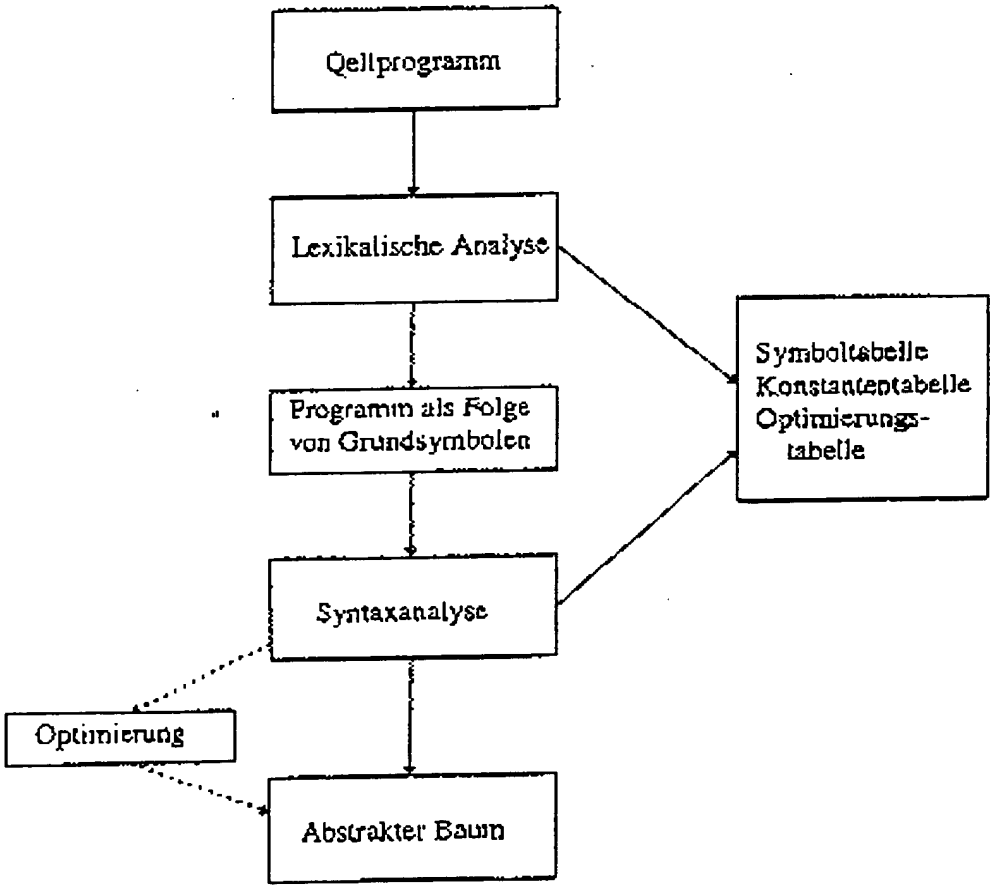


FIG. 1

Nummer: DE 196 22 365 A1
Int. Cl.⁶: G 06 F 5/00
Offenlegungstag: 11. Dezember 1997

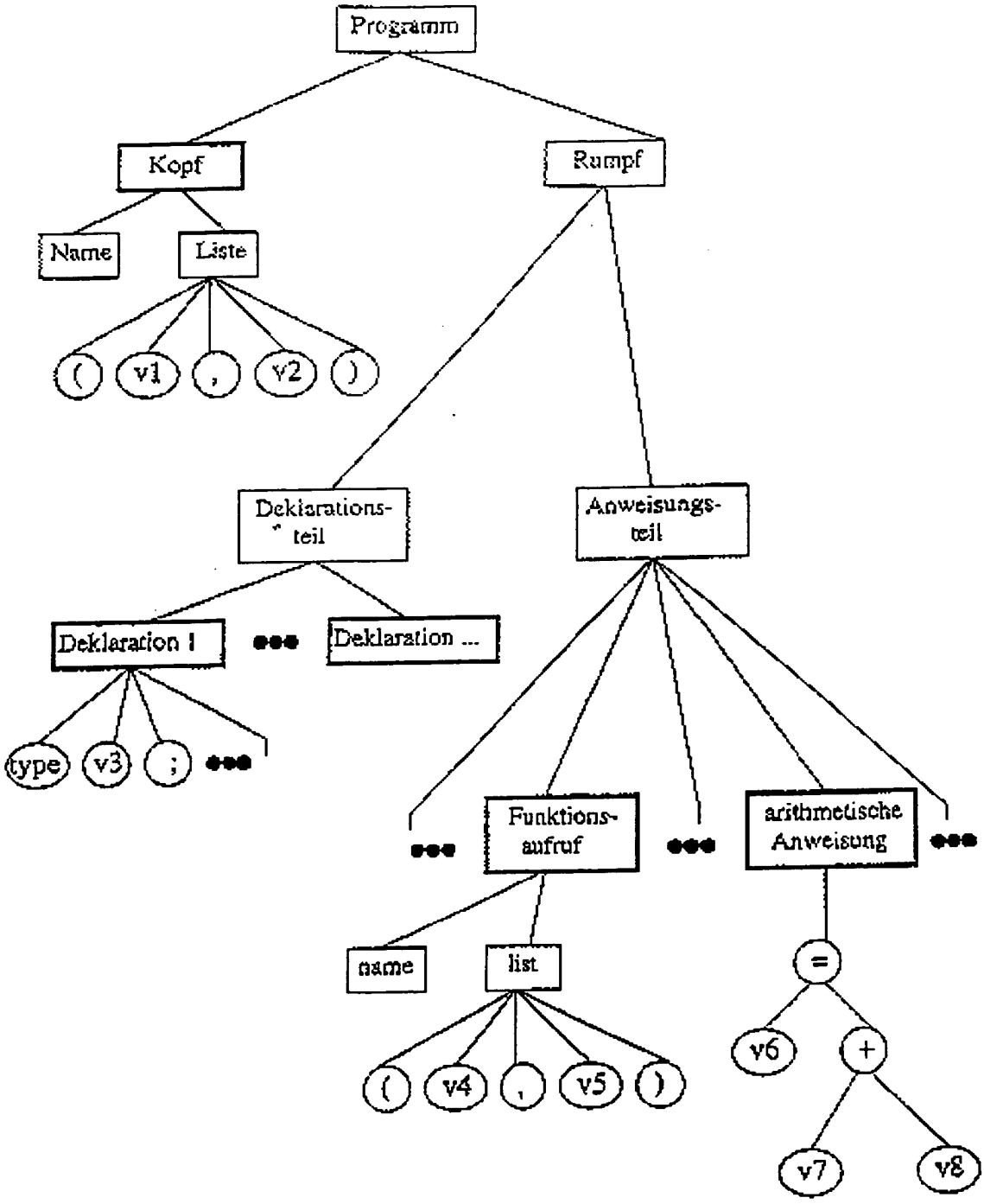


FIG. 2

Nummer: DE 196 22 365 A1
Int. Cl.⁶: G 06 F 5/00
Offenlegungstag: 11. Dezember 1997

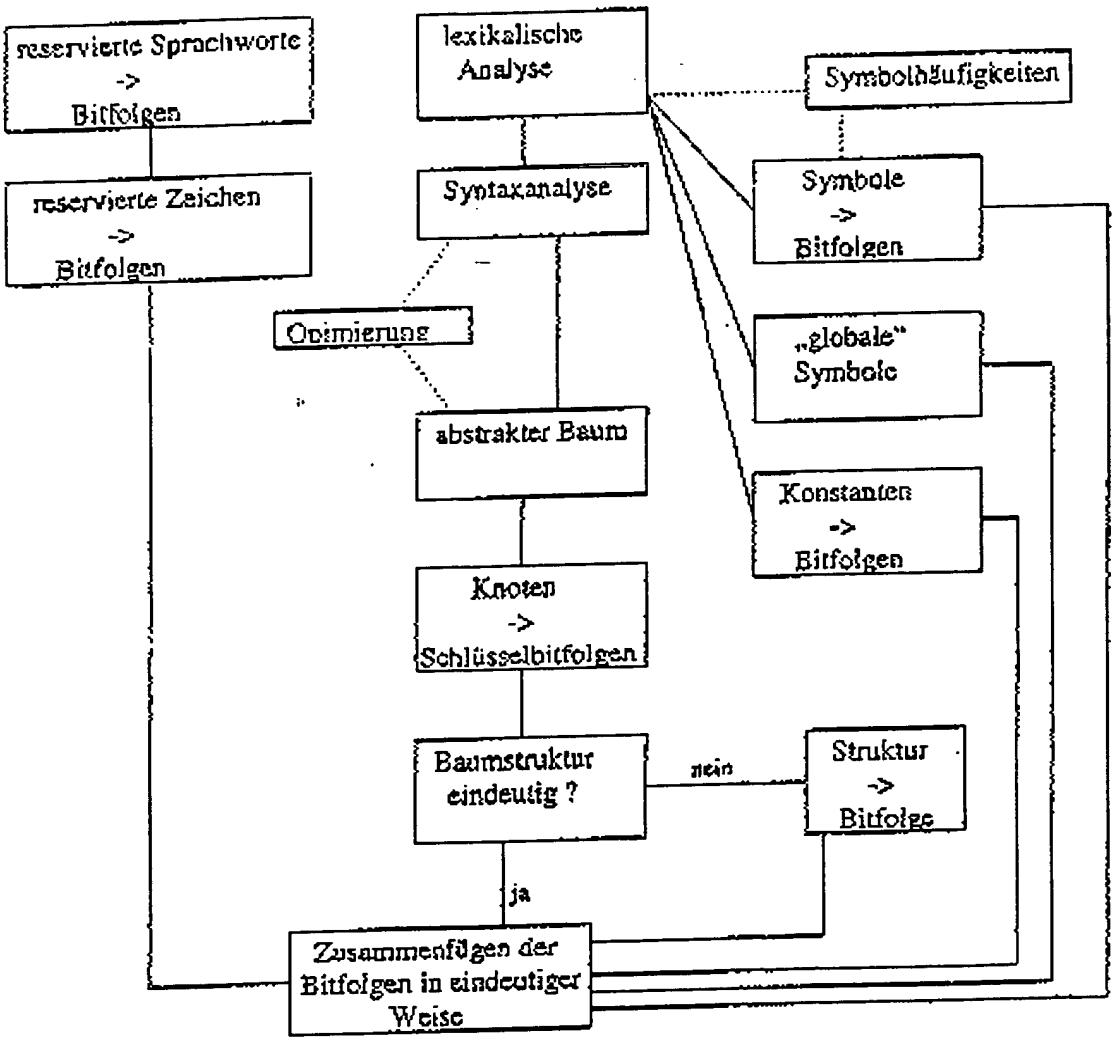


FIG. 3